Data Communication Error Measurement

This 8080 machine-language program checks for errors in data-transmission systems.

Terry A. Conboy
1231 Crestview Drive
San Carlos CA 94070

One of the more fascinating aspects of computing is data communications. Extensive networks link together commercial computers with satellite processors, remote terminals and even other large machines. The promise of electronic mail cannot be realized without an intricate web of data communication channels.

Computer hobbyists are only just beginning to see the potential of communication with the machines of other enthusiasts for bulletin boards, computer game competition, expanded processing capabilities and other imaginative uses.

Common to all users of data communication, whether commercial or hobbyist, is the need to maintain the reliability of the transmission channels and data communications equipment. Developers of modems must be able to verify the performance of their designs in some quantitative way. The technique presented here will allow the hobby computer user to make the measurements necessary to perform both of these functions.

There is a large variety of commercial test equipment available to check data communications links. Almost all of the units carry price tags larger than the total investment by an average hobbyist. Fortunately, there is a relatively simple way to generate sequences of bits that can be inspected by a digital circuit or computer to ferret out errors caused by noise, distortion, interference or other mischief in the transmission path.

My first thought was to build a simple version of a stand-alone test box with the capability of sending and checking a stream of bits for errors in transmission. Having almost completed the design of this error-rate test box, I figured that it would be worthwhile to parallel the development of the hardware with a program for my computer system to perform the same function. It didn't take long to realize that the software solution was far easier to accomplish, and I soon completely forgot about the hardware approach.

Error-Checking Methods

Several approaches could be taken to measure the error performance of a data communications system. Use of a cyclic redundancy check (CRC) is not specific enough. It lets you know that there is an error in a block transmission, but there could easily be more than one.

A parity bit for each character could be generated and inspected. This has a few problems. Multiple errors can easily cancel out. If the transmission channel is really in bad shape, such as is often the case on HF radio circuits, characters may be missed entirely, along with the parity bit that is supposed to check them. Certain codes, such as the five-level code (Murray), do not provide for parity bits at all.

The "quick brown fox" that has been jumping over lazy dogs for many decades could be used to spot erroneous characters. Again, multiple errors per character could not be easily detected. It would also seem that synchronization at the receiving end could be a problem with the fox test sequence.

Alternating characters containing reversing bit patterns, such as the R-Y test used in five level, would allow bit error checking, but could fail to pinpoint errors that occur when other characters are transmitted.

By far the most common method, and the one with the fewest negative points, is the use of a *pseudorandom binary sequence (PRBS).* The PRBS can be generated and checked in hardware or software. It allows an accurate count of errors at the receiving end. Because of the variety of patterns of bits created, any sensitivity of the data communication equipment to particular patterns will be stimulated.

Generating a PRBS

The first item to consider is the method used to generate a PRBS. The usual approach is to use a long shift register with multiple feedback taps. The feedback taps are added *modulo 2*, and the result is stuffed back into the input of the shift register. This is shown for a four-stage shift register in Fig. 1.

There is no mystery to modulo 2 arithmetic. The result of a modulo 2 addition is obtained by adding the numbers in the normal fashion and then dividing by 2. The remainder is the sum mod 2. In a binary numbering system, this is really a snap. Just add all of the single bits (one-digit binary numbers) and ignore the carry. For adding only two bits, this is identical to the exclusive OR function. Still another way of generating the mod 2 sum is to take the odd parity of the bits to be added. If there are an odd number of 1s, the result is equal to 1. If there are an even number of 1s, the answer is 0.

For the four-stage shift register shown, there are fifteen different states. You might expect there would be sixteen, which is the number of different combinations of four bits. Unfortunately, when all of the registers contain 0, the PRBS generator will latch up, since the feedback will always be 0. Whether the sequence is generated in software or hardware, this all 0 condition must be checked for to avoid initialization problems.

In general, the number of states that can occur until the sequence repeats is $(2^n - 1)$, if the proper taps are chosen for the (n) stages. Table 1 indicates the positions of the taps and the sequence length for shift registers up to sixteen stages. There are other combinations of taps that will produce maximal-length sequences, but those given involve the minimum number of taps. "Pseudorandom Sequences and Arrays" (F. Jessie

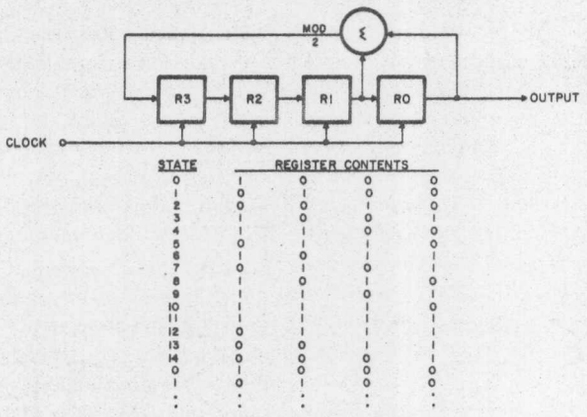

Fig. 1. A four-stage pseudorandom binary sequence generator. The contents of the registers change state on one edge of the clock. In this case, the modulo 2 summer could be replaced by an exclusive OR gate.

MacWilliams and Neil J.A. Sloane, *Proceedings of the IEEE*, December 1976, p. 1715) gives a complete discussion of PRBS generators and an extensive bibliography.

Checking for Errors

Although the sequences generated by the feedback shift registers can be extremely long, they are also very predictable. This is what makes them useful for error checking. My first reaction to the concept made me wonder if it was necessary for the error detection logic to scan the entire sequence before the detector could "find its place."

Fortunately, this isn't the case. The detector circuit must only receive a number of correct bits equal to the number of stages in the PRBS generator before it can predict exactly which binary value the next bit should be. If there is a disagreement, the new bit is in error.

An error detector for a PRBS generated by a four-bit shift register is shown in Fig. 2. The error detector must have the same number of stages as the generator. It must also have the same configuration of feedback taps. The difference between the generator and the error detector is that there is no actual feedback in the detector. The modulo 2 sum of the feedback taps is merely compared to the bit, which will be shifted into the register next. This should be the same bit that was shifted into the generator shift register several time states previously.

The exclusive OR gate functions as a data comparator. If the incoming bit differs from the sum of the taps, the XOR gate output will go to a 1, indicating an error.

Remember that the detector shift register must be full of correct bits before an erroneous bit can be detected. Complications arise when the bad bit is shifted into the shift register on the following clock pulses. Eventually this bit gets to the taps and causes an incorrect sum to be formed. The result is that the new incoming bit is declared to be wrong even though it is actually right. For isolated bit errors, there will be an error noted for each tap, plus the one for the first time it is found. In the case of the four-stage detector in Fig. 2, there will be three errors counted for each bad bit received.

This becomes further complicated if the errors occur in bursts. You can imagine what will happen if another error comes along just at the same time a previous error is in position at the taps. Two wrongs seem to make a right. In practice this will cause the error total to be slightly smaller than you would have expected. Unless the data communications channel that you are monitoring is really rotten, the underestimate of errors won't be large.

Use in Asynchronous Systems

What we have been considering is an oversimplified data transmission system. There has been no mention of the source of the clocks for the two shift registers. Obviously, the clock at the transmitting end sets the baud rate for the system. The clock at the receiving end must be magically regenerated from the transitions of the incoming bits or must be sent in some other devious fashion from the transmitting end, perhaps by a separate channel. Regeneration of the received clock is a normal function in a synchronous data transmission arrangement, but not many hobbyists are using synchronous systems. For that reason, we'll restrict our attention to asynchronous schemes.

At first glance, using an asynchronous transmission system might seem to present some problems. What do you do about the required start bit, parity bit (if used) and stop bit(s)? Fortunately, normal methods of sending serial data take care of all of these details. A device such as a UART (universal asynchronous receiver-transmitter) is typically used in a serial I/O port.

At the transmitting end, the desired number of the least significant bits of the PRBS shift register are parallel loaded into the UART for transmission. The UART handles the addition of the overhead bits required for transmission. At the receiving end, it takes care of character-by-character synchronization and removes all of the added bits. The parallel output of the UART is then shifted into the error detector one bit at a time. This can be done at any convenient rate as long as it is fast enough to be completed before the next character arrives.

Put another way, we pretend that groups of bits from the PRBS make up the information bits of asynchronous characters. We end up sending the bit sequence a bunch at a time without mixing up the original order.

A note of caution is worth mentioning: even though the sequence of bits from the shift register at the sending end is

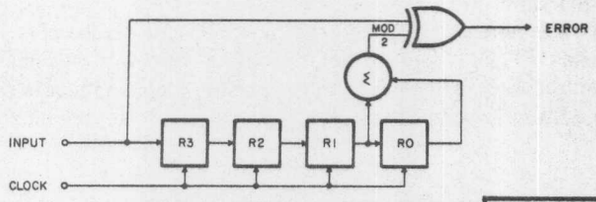

Fig. 2. A four-stage PRBS error-checking circuit. There is no actual feedback as in the generator. The modulo 2 sum of the taps is compared with the incoming bit by the exclusive OR gate. An output of 1 indicates an error. Here again, the summer could be an exclusive OR gate.

Number of Stages	Feedback Taps On Registers	Sequence Length
2	1,0	3
3	1,0	7
4	1,0	15
5	2,0	31
6	1,0	63
7	1,0	127
8	6,5,1,0	255
9	4,0	511
10	3,0	1023
11	2,0	2047
12	7,4,3,0	4095
13	4,3,1,0	8191
14	12,11,1,0	16383
15	1,0	32767
16	5,3,2,0	65535

Table 1. Required feedback taps and resulting pseudorandom binary sequence length for shift registers from two to 16 stages.

Program listing.

```
                        Save registers on stack
                PUSH B
                PUSH D
                PUSH H
                PUSH PSW
0000 C5 COMTST
0001 D5
0002 E5
0003 F5
```

pseudorandom, the stream of characters that results is not really random, as might be judged by certain mathematical standards. This isn't the best way to implement a random number generator for gambling purposes.

Program Description

The error measurement program is written in 8080 machine language. I'm sure that with suitable cleverness, it could be done in BASIC, but the shift and rotate instructions of the processor are ideally suited to the generation and checking of the bit sequences. Why make it hard?

A 15-stage configuration was chosen for the PRBS generator and error detector. This provides a long sequence without creating a software mess. A 16-stage design would have required the simulation of two more taps and would have complicated the error-checking arrangement. The varied 16-bit manipulations in the 8080 instruction set simplify the program greatly.

A flowchart of the program appears in Fig. 3. There are three main sections in the program. The control section takes care of initialization, reset of the error count and exit from the routine. The transmission section implements the PRBS shift register and sends the individual characters to the modem I/O port. The receiving section gets characters from the modem I/O port, shifts the bits through the error-checking shift register and keeps and finally outputs the total of any errors found.

By far the most confusing aspects of the program are the PRBS generator and the error-checking section. Both the transmit and receive shift register contents are kept in RAM between generation and checking of the bits in a character.

In the transmitting section of the program, the shift register contents are transferred to the HL register pair. There is a check for all 0s to avoid a latch-up state. The contents of the L register are then moved to the accumulator, and all but the two least significant bits are masked off. The parity flag is then checked. If the parity is odd, then the feedback is 1. To stuff the feedback into the shift register, the H register is ORed with 80 hex to set the most significant bit to 1. This step is skipped when the feedback is 0.

After the feedback is added, the H and L registers are rotated (in the accumulator) through the carry bit. This permits the bit that falls out of the H register to be moved into the L register. This calculation of feedback and shifting to the right is repeated once for each of the information bits to be sent in the character (7 for ASCII).

The contents of the HL regis-

Fig. 3. The flowchart of the error-rate measurement program.

```
0004  AF              XRA A
0005  32 05 01        STA TXTCNT   Zero Text Counter
0008  CD D8 00        CALL RESET
000B  DB 03    CONTRL IN KBSTAT    Get Keyboard Status
000D  E6 02           ANI 02H      Bit 1 = Data Available
000F  CA 25 00        JZ CKCNT     If no data, bypass decoding
0012  DB 02           IN KBDATA    Get Keyboard Data
0014  E6 7F           ANI 7FH      Ignore any Parity Bit
0016  FE 03           CPI 03H      Is it "CONTROL-C"?
0018  C2 20 00        JNZ CONT     No, continue program
001B  F1              POP PSW      Yes, return registers
001C  E1              POP H
001D  D1              POP D
001E  C1              POP B
001F  C9              RET          and exit...
0020  FE 20    CONT   CPI 20H      Is it a "SPACE"?
0022  CC D8 00        CZ RESET     Yes, reset Error & Char Counters
0025  2A 06 01 CKCNT  LHLD CHCNT   Get Character Counter from RAM
0028  7C              MOV A,H
0029  B5              ORA L        Is Counter zero?
002A  CC D8 00        CZ RESET     Yes, reset Error & Char Counters
002D  0E 07           MVI C,07H    Set up no. of Information Bits
002F  2A 08 01        LHLD TSR     Get Transmit Shift Reg from RAM
0032  7C              MOV A,H
0033  B5              ORA L        Shift Reg contents all zeroes?
0034  C2 38 00        JNZ OK       No, do nothing
0037  23              INX H        Yes, bump register to unlatch
0038  7D        OK    MOV A,L
0039  E6 03           ANI 03H      Select Feedback Taps
003B  7C              MOV A,H
003C  EA 41 00        JPE FBZ      If Parity Even, Feedback is zero
003F  F6 80           ORI 80H      If Odd, set Feedback to one
0041  1F        FBZ   RAR          Shift Feedback into higher bits
0042  67              MOV H,A
0043  7D              MOV A,L      Transfers shift out via Carry
0044  1F              RAR          Shift lower order bits
0045  6F              MOV L,A
0046  0D              DCR C        Decrement Info Bit Counter
0047  C2 38 00        JNZ OK       Loop until all bits shifted
004A  22 08 01        SHLD TSR     Keep Shift Reg contents in RAM
004D  DB 01    CHSEND IN MDSTAT    Get Modem UART Transmit Status
004F  E6              ANI 01H      Bit 0 = Transmit Buffer Empty
0051  CA 4D 00        JZ CHSEND    Wait until empty
0054  7D              MOV A,L      Get low order bits of Shift Reg
0055  D3 00           OUT MODEM    Send the new Character formed
0057  0E 07           MVI C,07H    Set up no. of information bits
0059  06 00           MVI B,00H    Clear Char Error Counter
005B  2A 0A 01        LHLD RSR     Get Rcvr Shift Reg contents
005E  DB 01           IN MDSTAT    Get Modem UART Receive Status
0060  E6 02           ANI 02H      Bit 1 = Receive Data Available
0062  CA 0B 00        JZ CONTRL    Go to pgm beginning if no char
0065  DB 00           IN MODEM     Get new Received Character
0067  00              NOP          Room to output the character
0068  00              NOP           to your front panel...
0069  0F        AGN   RRC          Rotate LSB to MSB
006A  57              MOV D,A      Save rotated character
006B  E6 80           ANI 80H      Mask off all but MSB
006D  5F              MOV E,A      Save incoming bit
006E  B4              ORA H        Combine w/MSD of Rcv Shift Reg
006F  67              MOV H,A
0070  7D              MOV A,L      Get LSD of Rcv Shift Reg
0071  E6 03           ANI 03H      Select Feedback Taps
0073  B3              ORA E        Combine with incoming bit
0074  EA 7C 00        JPE GOOD     If Parity even, Bit is correct
0077  3E 01           MVI A,01H    If odd, set A to 01
0079  80              ADD B         and add to Char Error Count
007A  27              DAA             in BCD
007B  47              MOV B,A
007C  7C        GOOD  MOV A,H
007D  1F              RAR          Shift higher order bits
007E  67              MOV H,A
007F  7D              MOV A,L
0080  1F              RAR          Shift lower order bits
0081  6F              MOV L,A
0082  7A              MOV A,D      Restore rotated Rcvd Character
0083  0D              DCR C
0084  C2 69 00        JNZ AGN      Repeat until all bits checked
0087  22 0A 01        SHLD RSR     Restore Rcv Shift Reg in RAM
```

ter pair are moved back to RAM, and the required least significant bits are sent as a serial character. Actually, eight bits are sent to the UART, and the way it is configured determines how many bits will be used in the character sent.

In the receiving section of the program, the least significant bit of the received character, the bit to be checked, is moved into the most significant bit of the E register. After the two least significant bits of the shift register contents are isolated by masking, the E register is ORed with the accumulator to leave an eight-bit number with only three active bits. These three bits represent the two feedback taps and the incoming bit that is being checked. If the parity of the accumulator contents is even, the bit is good. If the parity is odd, the received bit is bad and the error count for that character is incremented (in BCD).

After each bit is checked, the receiving shift register, contained in the HL register, is shifted. The checking and shifting are repeated until all of the information bits are checked. Then the error total is updated.

The error count is output as "BER/10↑4 = XXXX." This is a cryptic way of saying that the bit error rate per 10^4 bits is XXXX. In actuality, 10,000 bits were not really checked. Remember that each error going through the receiving shift register produces three counts. For this reason, only 3333 bits need to be checked. For ASCII characters with seven information bits, this represents 476 characters. In hexadecimal this is 1DC. This is the value to which the character count is reset for each measurement cycle.

To run the program, you must call it from your operating system or another program. The program saves all of the 8080 registers and restores them when the test program is terminated. The only assumption made by the program is that the stack pointer has been properly initialized. Space for at least 14 bytes on the stack must be available.

Pressing the space bar on your keyboard during execution will reset the error total and initiate a new measuring cycle. This is useful to clear the count after the data path has been interrupted. This includes any time that the program is called, since the receiving register will take two or three characters to properly synchronize with the incoming PRBS.

Note that the program will continue to send characters as long as it is executing. The error count will be output only during times that characters are being received. If the program is to be used for receive-only measurements, such as in half-duplex data systems or tape applications, the transmit serial output can be ignored.

To terminate the operation of the test program, type "Control-C" (the end-of-text character). The routine will "pop" all of the original contents of the 8080 registers and will return to the calling program.

Port	Direction	Function of Port	Status Bit Information
00	Out	Data to Modem	
00	In	Data from Modem	
01	In	Status of Port 00	b0-Output Buffer Empty;b1-Input Data Ready
02	In	Data from Keyboard	
03	In	Status of Port 02	b1-Keyboard Data Ready

Table 2. Input/output port assignments used in the program. Use of status bits is indicated.

Program Modifications

It is extremely unlikely that the program as listed will run on anybody's system but mine. My system is Z-80 based and has two RS-232C serial I/O ports. One of them is used to interface a serial keyboard. The other is used to connect to a modem or other communications device. The measurement of error rate is displayed via a DMA video interface using a standard driver routine that is part of my system monitor program.

The addresses of the I/O ports in my system and the associated status bits are given in Table 2. Besides modifying the I/O assignments, the only other change that should be necessary is to supply the address of the routine that you use to send data to your CRT. Although my CRT is driven by a shared-memory interface, any output device is usable. The output device must be at least as fast as the system being tested. For instance, a 1200 baud CRT terminal would do nicely if the data

```
008A 2A 0C 01          LHLD ERRCNT  Get error total from RAM
008D 78                MOV A,B      Contains Char Error Count
008E 85                ADD L        Add LSD Error Counter
008F 27                DAA          in BCD
0090 6F                MOV L,A
0091 3E 00             MVI A,00H    Clear A but not Carry
0093 8C                ADC H        Add Carry & MSD of Error Count
0094 27                DAA          in BCD
0095 67                MOV H,A
0096 22 0C 01          SHLD ERRCNT  Return Error Total to RAM
0099 2A 06 01          LHLD CHCNT   Get Character Counter from RAM
009C 2B                DCX H        Decrement it
009D 22 06 01          SHLD CHCNT   And return to RAM
00A0 7C                MOV A,H
00A1 B5                ORA L        Check it for zero

00A2 C2 BF 00          JNZ DSPE     If not zero, bypass buffer load
00A5 21 00 01          LXI H,TBFR   Set up pointer to Text Buffer
00A8 3A 0D 01          LDA HIERCT   Get high byte of Error Count
00AB CD E5 00          CALL BCDS    Hex to ASCII & into Buffer
00AE 3A 0C 01          LDA ERRCNT   Get low byte of Error Count
00B1 CD E5 00          CALL BCDS    Hex to ASCII & into Buffer
00B4 3E 0F             MVI A,0FH    Initialize Buffer Counter
00B6 32 05 01          STA TXTCNT   and keep in RAM
00B9 21 F5 00          LXI H,MSG    Set pointer to message area
00BC 22 0E 01          SHLD TXTPTR  and keep in RAM
00BF 3A 05 01    DSPE  LDA TXTCNT   Get Text Counter from RAM
00C2 B7                ORA A        Check for zero
00C3 CA 0B 00          JZ CONTRL    If no more, back to beginning
00C6 3D                DCR A        If more text, reduce Counter
00C7 32 05 01          STA TXTCNT   and put back in RAM
00CA 2A 0E 01          LHLD TXTPTR  Get Text Pointer
00CD 7E                MOV A,M      Get Message Char pointed to
00CE CD 03 F8          CALL CHD     and have it displayed on CRT
00D1 23                INX H        Bump pointer value
00D2 22 0E 01          SHLD TXTPTR  and return it to RAM
00D5 C3 0B 00          JMP CONTRL   Go back to beginning for more
00D8 21 DC 01    RESET LXI H,01DCH  No. of char per measurement
00DB 22 06 01          SHLD CHCNT   to initial value in Counter
00DE 21 00 00          LXI H,0000H  Zero value
00E1 22 0C 01          SHLD ERRCNT  put into Error Counter
00E4 C9                RET
00E5 47          BCDS  MOV B,A      Temporarily hold number
00E6 0F                RRC          Rotate
00E7 0F                RRC            to
00E8 0F                RRC              get the
00E9 0F                RRC                higher hex digit
00EA CD EE 00          CALL ASCII   Convert hex digit to ASCII
00ED 78                MOV A,B      Now do lower hex digit
00EE E6 0F             ANI 0FH      Select lower four bits
00F0 F6 30       ASCII ORI 30H      Make it an ASCII numeral
00F2 77                MOV M,A      Place in buffer area
00F3 23                INX H        Bump pointer value
00F4 C9                RET
00F5 0D          MSG   ASCII        Carriage Return
00F6 0A               ASCII        Line Feed
00F7 42               ASCII        'B'
00F8 45               ASCII        'E'
00F9 52               ASCII        'R'
00FA 2F               ASCII        '/'
00FB 31               ASCII        '1'
00FC 30               ASCII        '0'
00FD 5E               ASCII        '↑'
00FE 34               ASCII        '4'
00FF 3D               ASCII        '='

0100 30          TBFR  ASCII        '0' Will be changed by the
0101 30               ASCII        '0'  program to reflect the
0102 30               ASCII        '0'  number of errors found
0103 30               ASCII        '0'  in the measuring cycle
0104 00               NOP
0105 00          TXTCNT DATA       Keeps count of text left to go
0106 DC 01       CHCNT  DATA       Keeps count of char left to go
0108 55 55       TSR    DATA       Contents of Transmit Shift Reg
010A 55 55       RSR    DATA       Contents of Receive Shift Reg
010C 00          ERRCNT DATA       Low byte of Error Count Total
010D 00          HIERCT DATA       High byte of Error Count Total
010E F5 00       TXTPTR DATA       Pointer to Message & Text Bufr
0110 00                 NOP
```

Number of Information Bits Used	Constant at 002E and 0047	Two-byte constant at 00D9-00DA for an equivalent measurement of about				Time for a 10^4 bit measurement cycle seconds (with bps)
		10^3 Bits	10^4 Bits	10^5 Bits	10^6 Bits	
5	05	0043	029B	1A0B	FFFF*	110 (45.5)
6	06	0038	022C	15B4	D904	41 (110)
7	07	0030	01DC	129A	BA03	16 (300)
8	08	0024	01A1	1047	A2C3	3.5 (1200)

* Will give results about 1.7 percent too low. Length of measurement period is over three hours at 45.5 bps!

Table 3. Hexadecimal constants to be changed to give proper operation with a different number of information bits and with different measuring intervals. Approximate measurement cycle times are given for various data rates. Constants used in program are indicated in boxes.

link being measured were running at 1200 baud or less.

In addition to modifying the program to fit your system, you may want to change the equivalent number of bits over which the error rate is measured. For systems which are virtually error-free, a greater number of bits should be checked to give a higher probability of finding erroneous bits. Fewer bits per measuring cycle are desirable when system reliability is poor or to shorten the cycle time when making adjustments to improve performance.

Table 3 gives the constants that must be changed in the program to vary the number of bits per measurement cycle. Values for characters with five to eight information bits are included. An indication of the cycle time is given for some typical data rates. The values in boxes are those from the original program.

Applications

A common use of the program is to send test sequences over a loopback. Several different arrangements are shown in Fig. 4. Note that the loopback may be for digital signals, such as RS-232, directly at the computer or at the interface to the terminal equipment at the other end, or it may be for analog signals at either the sending modem or at the far end of the transmission path.

Some commerical modems have the capability of being looped-back at various points under control from a remote location. This permits maintenance to be done without requiring manual patching of signals. The part of the system causing a problem can easily be isolated with this technique.

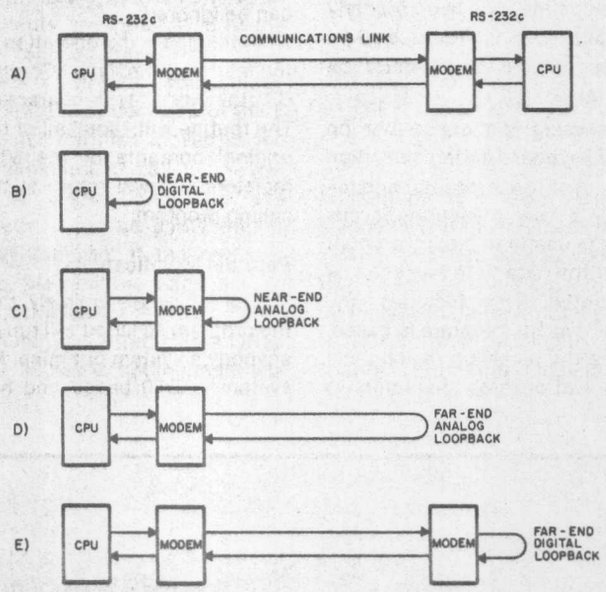

Fig. 4. Equipment arrangements for testing with the program. With the program executing in two communicating computers, configuration a can be used. Testing from one end only is possible with digital or analog loopbacks at various points as in b, c, d or e.

Of course, tests between two computers can be done if both of the processors are executing the program at the same time. One advantage of this arrangement is that it permits finding sources of errors that occur for transmissions in one direction only. These maybe found by loopback tests, but you will have trouble deciding which half of the loop is the culprit. Testing between two computers is also necessary for systems that are half duplex by nature. Radio channels in the HF spectrum are normally only one direction at a time on the same frequency and must be tested this way.

Another possible use of error-rate testing is to permit adjusting tape interfaces for data recording. The PRBS is first recorded and is then played back to allow counting errors that may be occurring.

Final Comments

I hope that this technique for measuring errors is helpful to hobby users of data communications. Even if you think my code is inefficient (which is probably the case), you should be able to use the ideas that are presented, even on other processors.

This method is another simple trick that the commercial people take for granted. As a result, hobbyists never find out the secrets that are "obvious" to the pros. ■

MOVING?

Let us know 8 weeks in advance so that you won't miss a single issue of *Kilobaud Microcomputing*. Attach old label where indicated and print new address in space provided. Also include your mailing label whenever you write concerning your subscription. It helps us serve you promptly.

☐ Address change only
☐ Extend subscription
☐ Enter new subscription
☐ 1 year $18.00
☐ Payment enclosed
☐ Bill me later

AFFIX LABEL

If you have no label handy, print OLD address here.

Name _____
Address _____
City _____ State _____ Zip _____

print NEW address here:

Name _____
Address _____
City _____ State _____ Zip _____

Kilobaud **Microcomputing** P.O. Box 997 ● Farmingdale NY 11737